Solving Fokker-Planck equation using deep learning

Yong Xu, Hao Zhang, Yongge Li, Kuang Zhou, Qi Liu, and Jürgen Kurths

COLLECTIONS

Paper published as part of the special topic on When Machine Learning Meets Complex Systems: Networks, Chaos and Nonlinear Dynamics

Note: This paper is part of the Focus Issue, "When Machine Learning Meets Complex Systems: Networks, Chaos and Nonlinear Dynamics."

View Online   Export Citation   CrossMark

ARTICLES YOU MAY BE INTERESTED IN

Causal network discovery by iterative conditioning: Comparison of algorithms
Chaos: An Interdisciplinary Journal of Nonlinear Science **30**, 013117 (2020); https://doi.org/10.1063/1.5115267

The maximum likelihood climate change for global warming under the influence of greenhouse effect and Lévy noise
Chaos: An Interdisciplinary Journal of Nonlinear Science **30**, 013132 (2020); https://doi.org/10.1063/1.5129003

Analytical techniques for linear response formula of equilibrium states
Chaos: An Interdisciplinary Journal of Nonlinear Science **30**, 013134 (2020); https://doi.org/10.1063/1.5122810

Solving Fokker-Planck equation using deep learning

Yong Xu,[1,2,a)] Hao Zhang,[1,3] Yongge Li,[4] Kuang Zhou,[1] Qi Liu,[1] and Jürgen Kurths[5,6]

AFFILIATIONS

[1] Department of Applied Mathematics, Northwestern Polytechnical University, Xi'an 710072, China
[2] MIIT Key Laboratory of Dynamics and Control of Complex Systems, Northwestern Polytechnical University, Xi'an 710072, China
[3] Department of Engineering Mechanics, Northwestern Polytechnical University, Xi'an 710072, China
[4] Center for Mathematical Sciences and School of Mathematics and Statistics, Huazhong University of Science and Technology, Wuhan 430074, China
[5] Potsdam Institute for Climate Impact Research, Potsdam 14412, Germany
[6] Department of Physics, Humboldt University Berlin, Berlin 12489, Germany

Note: This paper is part of the Focus Issue, "When Machine Learning Meets Complex Systems: Networks, Chaos and Nonlinear Dynamics."
[a)]**Author to whom correspondence should be addressed:** hsux3@nwpu.edu.cn

ABSTRACT

The probability density function of stochastic differential equations is governed by the Fokker-Planck (FP) equation. A novel machine learning method is developed to solve the general FP equations based on deep neural networks. The proposed algorithm does not require any interpolation and coordinate transformation, which is different from the traditional numerical methods. The main novelty of this paper is that penalty factors are introduced to overcome the local optimization for the deep learning approach, and the corresponding setting rules are given. Meanwhile, we consider a normalization condition as a supervision condition to effectively avoid that the trial solution is zero. Several numerical examples are presented to illustrate performances of the proposed algorithm, including one-, two-, and three-dimensional systems. All the results suggest that the deep learning is quite feasible and effective to calculate the FP equation. Furthermore, influences of the number of hidden layers, the penalty factors, and the optimization algorithm are discussed in detail. These results indicate that the performances of the machine learning technique can be improved through constructing the neural networks appropriately.

The Fokker-Planck (FP) equations, as an indispensable part of stochastic dynamics, have been widely used in many different fields, such as physics, chemistry, and biology. The probability density functions can be obtained from the associated FP equations, which play an extremely important role in stochastic resonance, mean first passage time, engineering reliability, etc. Generally speaking, however, the exact solutions of the FP equations can only be obtained under some strict conditions. Thus, a number of numerical methods such as the finite element method and the path integral technique have been proposed to solve the FP equations. These methods inevitably require mesh or associated transformations, which increase the amount of computation and operability. The problem becomes worse when the system dimension increases. Only recalculation or interpolation can be used when it is necessary to solve the value of points not on the grid. In this paper, we develop a novel deep learning method that well fits the nonlinear function to solve the FP equation. We add the normalization condition to avoid the trial solution to be zero. Meanwhile, we design a penalty factor for the loss functions to avert the appearance of a local optimum. Furthermore, a detailed algorithm procedure is described and employed to four illustrative examples including one- and two-dimensional systems. The results illustrate the validity and maneuverability of the developed machine leaning to solve the FP equation compared with the exact or numercial results.

I. INTRODUCTION

The FP equation describes the time evolution of the probability density functions (PDFs) of stochastic complex systems, which is

a quantitative equation changing a stochastic problem to a deterministic one. It is a central topic in stochastic issues due to its important relation to PDFs, stochastic resonance,[1] mean first pass time,[2] and diffusion. Besides, it has gained broad applications in statistical physics,[3] chemistry,[4,5] biology,[6,7] engineering,[8] economy, and finance.[9] Its exact stationary solution can only be obtained for a few conceptual models under very strict conditions, and it is difficult to find its exact stationary solution in general.[10] Therefore, several numerical techniques have been developed to achieve solving of the FP equations.[11–21]

In general, the PDFs of stochastic dynamics can be obtained from two main categories: One is directly to solve the FP equation through several techniques like the finite element method,[11,12] finite difference method,[13,14] path integral method,[15–17] variation method[18,19] and the Galerkin method.[20,21] These approaches focus on discretizing the computational domain into a set of grid points and solving approximate solutions on the grid, but the solution is only calculated at the grid point, and the evaluation of any other point requires an interpolation or other reconstruction techniques. For one-dimensional (1D) problems, these methods are very effective, but in two-dimensional (2D) and higher-dimensional systems, they consume much computing resources and even have troubles to overcome the curse of dimensionality. In addition, a sparsity of the grid seriously affects the accuracy of calculation, while a dense grid will inevitably lead to a sharp increase in the amount of calculation. The other approach is to solve the stochastic differential equations (SDEs) and then statistically calculate its transition probability density via the so-called Monte Carlo method.[22,23] Although this approach does not require interpolation, it needs a large number of sample paths and the accuracy of its solution is related to the amount of generated data. For a 2D system only, the computational cost of solving the joint probability density function (JPDF) may be too large to be obtained. Therefore, it is still a problem that needs to be solved to find an alternative technique without interpolation calculation and with rather low computational complexity. The use of artificial neural networks (ANNs) provides a promising way to solve this problem.

Actually, ANNs have attained considerable attention as robust and effective tools for the function approximation.[24,25] In 1989, Cybenko[26] theoretically proved that an ANN using sigmoid as the activation function could fit any continuous function over a compact subset of n-dimensional real space. A large number of scholars have focused on using ANNs to solve ordinary differential equations and partial differential equations due to the strong ability of ANNs to fit continuous functions.[27–35] Here, a supervised learning model was set to discretize the computation interval and boundary interval as the input of the network and took the form of differential equations and boundary conditions as the supervised conditions of the network training, so as to make the trial solution approach the true solution through a minimization criterion. From the perspective of the network construction, it can be divided into two forms: a single hidden layer neural network and a multiple hidden layer neural network. To estimate the partial derivatives in differential equations, the classification can be divided into three categories. A single hidden layer neural network is used to directly apply the derivative formula,[27–31] while others[32,33] constructed the network using a multilayer neural network with an automatic differential technique to estimate the derivative, and in Ref. 35 the Monte Carlo method is taken to estimate the derivatives under the guidance of Galerkin's idea. To summarize, solving the differential equations with the ANN technique offers the following preferences in comparison with classical numerical schemes: (1) ANNs learn the analytic solution, (2) the solution search does not require any coordinate transformation, (3) as the number of sampling points increases, the computational complexity does not increase rapidly, and (4) one can calculate the value of the solution quickly.

The FP equation is a kind of second-order parabolic partial differential equation, but the above methods cannot be applied to solving the FP equation. There are two main reasons: first, there is no constant term in the format of FP equation and the boundary condition is zero, which makes zero become the trial solution during the training which is not the exact solution. So, it is necessary to increase the supervision condition to solve this problem. The second is that the traditional mean square error loss function cannot be used to train the network effectively due to the increased monitoring conditions, so it is necessary to change the format of the loss function.

In this paper, we propose a method applying deep learning to solve the FP equation to overcome the mentioned points. In addition to take the form of the equation and the boundary condition as the supervision condition, we will also take the normalization condition as the supervision condition. Hence, the solution of the FP equation is a PDF or JPDF, which can avoid the situation that the trial solution is zero. In order to train the three components of the loss function well, a penalty factor is introduced to solve this problem.

The structure of this paper is as follows: Sec. II gives a detailed description of the method to solve a FP equation using deep learning which is called the Deep learning-FP (DL-FP) method, and the proposed algorithm scheme is presented. In Sec. III, four examples consisting of 1D, 2D, and three-dimensional (3D) systems are analyzed by means of the DL-FP method, and the results are discussed. In Sec. IV, we will discuss three concerns in the algorithm procedure: (1) the effect of the number of hidden layer on the calculated result, (2) the importance of the penalty factor in the algorithm, and (3) how to select the optimization technique in the DL-FP algorithm. Finally, conclusions are presented to close this paper.

II. METHODOLOGY

A. The FP equations

Consider the following general n-dimensional SDE:

$$\dot{X} = f(X) + \Lambda \Gamma, \tag{1}$$

where $X = [x_1, x_2, \ldots, x_k, \ldots, x_n]^T$ is an n-dimensional variable ($n \geq 1$), Λ is a square matrix which is assumed here to be constant, Γ is a vector of uncorrelated standard Gaussian white-noise processes, and $f(X)$ is a general n-dimensional vector function of X. For Eq. (1), the solution process is still a Markov one and the joint PDF (JPDF) of the stationary response satisfies the reduced FP equation as follows:

$$-\sum_{k=1}^{n} \frac{\partial}{\partial x_k} \left[f_k(X)p(X) \right] + \sum_{j=1}^{n} \sum_{k=1}^{n} \frac{\partial^2}{\partial x_j \partial x_k} \left[\frac{m_{jk}(X)}{2!} p(X) \right] = 0, \tag{2}$$

where $m_{ij}, i,j = 1,2,\ldots,n$ are the elements of the matrix M given by $M = 2\pi \Lambda \Lambda^{\mathrm{T}}$. $p(X)$ is the transition PDF of Eq. (1), which must satisfies the normalization condition and $p(X)$ tends to zero as X tends to infinity.

In order to describe Eq. (2) expediently, we can rewrite it as

$$\mathcal{N}\left(p(X), p_X(X), p_{XX}(X)\right) = 0, \qquad (3)$$

where $\mathcal{N}[\cdot]$ is the FP differential operator.

B. The DL-FP algorithm

In this section, we will describe the numerical scheme to solve the FP equation via a deep ANN, and we name it as DL-PF algorithm.

Figure 1 illustrates a typical structure of an ANN which is also called deep neural network with L layers, where x is the input layer and y is the output one. $H^l = \left[h_1^l, h_2^l, h_3^l, \ldots, h_{n_l}^l\right]$ ($2 \leq l \leq L-1$) is the output of the lth hidden layer, and n_l is the number of nodes in each layer. $W = \left[w^1, w^2, \ldots, w^l \ldots, w^{L-1}\right]$ are the weights of network, and w^l represents the weight from the $(l-1)$th layer to the lth layer. The size of the matrix w^l is $n_{l-1} \times n_l$. $Bias = \left[b^1, b^2, \ldots, b^l \ldots, b^{L-1}\right]$ is the bias of the layers, and b^l represents the bias of the $(l+1)$th layer with the size $n_l \times 1$.

Then, we can describe a feed-forward process as

$$h_i^2 = F\left(w_i^1 x + b_i^1\right), \qquad (4)$$

$$h_i^l = F\left(\sum_{j=1}^{n_{l-1}} w_{ij}^{l-1} h_j^{l-1} + b_i^{l-1}\right), \quad 3 \leq l \leq L-1, \qquad (5)$$

$$y = \sum_{i=1}^{n_{L-1}} w_i^{L-1} h_i^{L-1} + b^{L-1}, \qquad (6)$$

where F represents the activation function (like a sigmoid, tanh, ReLU, etc.), and the output can be described as $y = \mathcal{F}(x;\theta)$, where

$\mathcal{F}$ is the composite function which is made of the activation function, and $\theta = [W, Bias]$ is denoted as the network parameters.

In general, the current methods solving the differential equations with ANN can be defined as follows: Assuming a differential equation with the initial condition or boundary condition to be

$$\begin{cases} \mathcal{D}(y(X)) = 0, & X \in \Omega, \\ \mathcal{B}(y(X)) = g(X), & X \in \partial\Omega, \end{cases} \qquad (7)$$

where $\mathcal{D}[\cdot]$ is the differential operator which contains the derivatives with respect to X and $\mathcal{B}[\cdot]$ is the boundary operator. The domain data $\mathcal{T}_D = \{X_i^D \in \Omega\}_{i=1}^{N_D}$ and the boundary data $\mathcal{T}_B = \{X_i^B \in \partial\Omega\}_{i=1}^{N_B}$ are discretized as input data of ANN, and the corresponding output is denoted as $\widehat{y} = \widehat{y}(X;\theta)$, where θ are the parameters of the underlying ANN. The loss function is denoted as $\mathcal{L} = E_1 + E_2$, and

$$E_1 = \frac{1}{N_D} \sum_{i=1}^{N_D} \left|\mathcal{D}\left(\widehat{y}\left(X_i^D;\theta\right)\right)\right|^2, \qquad (8)$$

$$E_2 = \frac{1}{N_B} \sum_{i=1}^{N_B} \left|\mathcal{B}\left(\widehat{y}\left(X_i^B;\theta\right)\right) - g\left(X_i^B\right)\right|^2, \qquad (9)$$

in which N_D and N_B represent the number of data in the training set and boundary one, respectively. Then, an optimization method is employed to train the network. This kind of training belongs to the supervised learning. The loss parts E_1 and E_2 guarantee that the training solutions satisfy the form of the given equation and the boundary conditions, respectively.

However, this method is not feasible to deal with the general FP equation (3). The training results will be zero with only the form of the equation and the boundary conditions. Therefore, we add the normalization condition as the supervision condition to avoid this confusion. At the same time, due to the addition of a supervision item, the training results in high-dimensional case will be invalidated. To address this problem, penalty factors will be introduced.
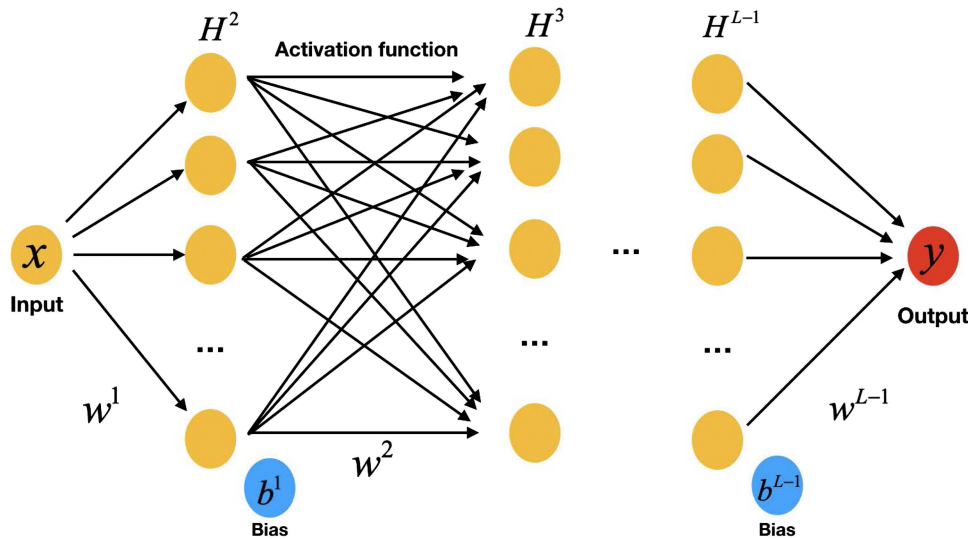

FIG. 1. The structure of a deep neural network.

Next, the detailed procedure of the proposed DL-FP algorithm will be described as follows.

(i) Step one is to select the training set. Since the input of an ANN should be a finite set, the range of independent variables of the FP equation needs to be truncated. Considering the FP equation with equilibrium states, the probability is zero when the independent variable is far from these states. So, we should select a sufficiently large interval. To be more accurate, we can use a Monte Carlo method to find the training interval.

After finding the training interval, we can get the training set $\mathcal{T}_D = \{X_i^D \in \Omega\}_{i=1}^{N_D}$. It is a discretization on each dimension of independent variables, and the step length is Δx. In the 1D case, we assume the interval to be $[a, b]$, then $N_p = (b - a)/\Delta x$, and the number of data in the training set is $N_D = N_p$. In the 2D case, we assume the interval to be $[a, b] \times [a, b]$ and the number of data in the training set is $N_D = (N_p)^2$. The boundary set is $\mathcal{T}_B = \{X_i^B \in \partial\Omega\}_{i=1}^{N_B}$, which describes the boundary of the training set. In the 1D case, the number of data in the boundary set is $N_B = 2$, and it is $N_B = 4(N_p - 1)$ in the 2D case. Both the training set and boundary sets are used as the inputs of the same ANN to get their output by a feed-forward neural network. At the same time, the absolute value function should be added on the outputs to ensure that the PDF is nonnegative.

(ii) Step two is to construct the loss function. To guarantee the obtained DL-FP solutions are the solutions of Eq. (3), three conditions should be satisfied. The first is the form of Eq. (3), the second is the normalization condition of PDF or JPDF, and the third is the boundary condition. For these purposes, we define the following loss function:

$$\mathcal{L} = \sum_{i=1}^{3} a_i \cdot E_i, \qquad (10)$$

where

$$E_1 = \frac{1}{N_D} \cdot \sum_{i=1}^{N_D} \left| \mathcal{N}\left(\widehat{p}\left(X_i^D | \theta \right) \right) \right|^2, \qquad (11)$$

$$E_2 = \left| \sum_{i=1}^{N_D} \Delta x^{Dim} \cdot \widehat{p}\left(X_i^D | \theta \right) - 1 \right|^2, \qquad (12)$$

$$E_3 = \frac{1}{N_B} \sum_{i=1}^{N_B} \left| \widehat{p}\left(X_i^B | \theta \right) \right|^2. \qquad (13)$$

Here, a_i, $i = 1, 2, 3$ represent the penalty factors, Dim denotes dimension of the variable X and $\widehat{p}(X|\theta)$ denotes the trial solution.

We find that Eq. (11) guarantees the approximate solution to satisfy the form of Eq. (3), Eq. (12) guarantees the normalization of PDF or JPDF, and Eq. (13) guarantees the boundary condition. These three supervisory conditions are indispensable, and the approximate solution obtained from DL-FP method will be zero if only the form of the function and the boundary condition are applied. Therefore, the normalization condition must be considered as a supervisory condition to avoid such situation that the approximate solution is zero.

(iii) Step three is to minimize the loss function $\mathcal{L}$. In this paper, we use the Adam method to minimize $\mathcal{L}$. We can also use the

---

Algorithm 1 DL-FP algorithm

Input: The training set $\mathcal{T}_D = \{X_i^D\}_{i=1}^{N_D}$, the boundary training set $\mathcal{B}_B = \{X_i^B\}_{i=1}^{N_B}$, the size of ANN.
Output: The DL-FP solution.

1: Initialize the weights and biases of the neural network based on the ANN size.
2: **for** iteration $j = 1, 2, ..., \mathcal{M}$ **do**
3:     Construct neural network only for training set and get the output $\widehat{p}(X|\theta_j)$.
4:     Take the first and second derivatives of the output $\widehat{p}_X(X|\theta_j)$, $\widehat{p}_{XX}(X|\theta_j)$.
5:     Construct the loss function.
6:     Update the neural network by minimizing the loss function

$$\mathcal{L} = a_1 \cdot \frac{1}{N_D} \cdot \sum_{i=1}^{N_D} \left| \mathcal{N}\left(\widehat{p}\left(X_i^D | \theta_j \right) \right) \right|^2$$
$$+ a_2 \cdot \left| \sum_{i}^{N_D} \Delta x^{Dim} \cdot \widehat{p}\left(X_i^D | \theta_j \right) - 1 \right|^2 + a_3 \cdot \frac{1}{N_B} \cdot \sum_{i=1}^{N_B} \left| \widehat{p}\left(X_i^B | \theta_j \right) \right|^2,$$

7:     Update the network:

$$\theta_{j+1} \leftarrow \theta_j + \delta\theta_j$$

8: **return** results

L-BFGS-B method,[36] and a comparison between the two methods will be discussed in Sec. IV C. Since there are three supervisory targets in the loss function, there will be a sequence in the optimization process. In order to avoid the loss function falling into the local minimum, we design a penalty factor to affect the process of supervisory training. Through our continuous experiments, it is found that the penalty factor can be taken as one in the 1D case. In the 2D case, the normalized supervision is easy to be satisfied. On the contrary, the form of Eq. (3) is not easy to be satisfied. So, we set the penalty factors as $a_1 = N_p$, $a_2 = 0.1$, $a_3 = 1$. The reason why we set the rules this way is that the training set increases in the 2D case. So, if the penalty factors of Eq. (11) increase, the error of each training point will be enlarged. Consequently, the supervisory condition of the form of function will be much accounted in the minimization procedure.

Algorithm 1 is the implemental process of the proposed DL-FP algorithm, and four different systems will be given to test the validity of our method in the next step.

III. EXAMPLES

The proposed framework in Sec. II provides a universal treatment of linear and nonlinear FP equations of fundamentally different nature, which are demonstrated by means of four examples including the 1D, 2D, and 3D systems.

A. Example 1

Consider a nonlinear system subject to a Gaussian noise as

$$\dot{x} = \alpha x - \beta x^3 + \sigma\Gamma \quad (\alpha > 0, \beta > 0), \qquad (14)$$

where Γ is a standard Gaussian white noise and σ is the noise intensity. The corresponding stationary FP equation is

$$-\frac{\partial}{\partial x}\left[(ax - \beta x^3)\, p(x)\right] + \frac{\sigma^2}{2}\frac{\partial^2}{\partial x^2}p(x) = 0. \qquad (15)$$

Then, its exact steady-state solution which is used to compare with the DL-FP solution is given by

$$p_s(x) = C \cdot \exp\left[\frac{1}{2\sigma^2}\left(2\alpha x^2 - \beta x^4\right)\right], \qquad (16)$$

where C is the normalization constant. Using the output $\widehat{p}(x|\theta)$ of the deep neural network to fit $p(x)$, we can construct the loss function as follows:

$$\mathcal{L} = \sum_{i=1}^{3} a_i \cdot E_i, \qquad (17)$$

where

$$E_1 = \frac{1}{N_D}\sum_{i=1}^{N_D}\left| -\frac{\partial}{\partial x_i^D}\left[\left(\alpha x_i^D - \beta\left(x_i^D\right)^2\right)\cdot\widehat{p}\left(x_i^D|\theta\right)\right]\right.$$
$$\left. + \frac{\sigma^2}{2}\cdot\frac{\partial^2}{\partial x_i^D}\widehat{p}\left(x_i^D|\theta\right)\right|^2, \qquad (18)$$

$$E_2 = \left|\sum_{i=1}^{N_D}\Delta x\cdot\widehat{p}\left(x_i^D|\theta\right) - 1\right|^2, \qquad (19)$$

$$E_3 = \frac{1}{N_B}\sum_{i=1}^{N_B}\left|\widehat{p}\left(x_i^B|\theta\right)\right|^2. \qquad (20)$$

To quantify the performance and accuracy of the DL-FP algorithm, we define

$$acc_{L_2} = 1 - \frac{\|\hat{y} - y\|_2}{\|y\|_2}, \qquad (21)$$

TABLE I. The accuracy of the DL-FP method for system (15).

Parameters	Loss value	acc_{L_2}	Iterations
$\alpha = 0.3, \beta = 0.5, \sigma = 0.5$	3.76×10^{-7}	0.9995	3×10^4
$\alpha = 0.5, \beta = 0.5, \sigma = 0.5$	8.33×10^{-7}	0.9988	3×10^4
$\alpha = 0.7, \beta = 0.5, \sigma = 0.5$	1.84×10^{-7}	0.9865	3×10^4

where $\hat{y}$ is the approximate solution, i.e., the DL-FP solution, and y is the exact solution, and $\|\cdot\|_2$ denotes the L_2 norm.

Based on the DL-FP method, we set the penalty factors to be $a_1 = 1$, $a_2 = 1$, $a_3 = 1$ and the training interval of the variable x to be $[-2.2, 2.2]$. We can get the training set and the boundary set with the step length of $\Delta x = 0.01$. Setting the size of the network with four layers and 20 nodes of each hidden layer and applying the Adam optimization technique, the minimum of the loss function is about 10^{-7} after 3×10^4 iterations. Figure 2 and Table I show the obtained results. The red dashed line is the DL-FP solution which is almost the same as the exact solution marked by the black solid line, and the average accuracy is 99.49%. These results indicate that the DL-FP method is highly effective in solving the FP equation.

B. Example 2

Consider the following SDE driven by an additive Gaussian noise:[37,38]

$$\dot{x} = f(x) + \sigma\Gamma, \qquad (22)$$

where $f(x) = \alpha - x + \frac{x^8}{1+x^8}$, Γ is a standard Gaussian white noise, and σ is the noise intensity. The corresponding stationary FP equation is

$$-\frac{\partial}{\partial x}\left[f(x)\cdot p(x)\right] + \frac{\sigma^2}{2}\frac{\partial^2}{\partial x^2}p(x) = 0. \qquad (23)$$

In order to check the validity of the DL-FP method, the Monte Carlo solutions are also given which are shown in Fig. 3.

In the calculation, the training interval of the variable x is selected to be $[-2, 4]$, then $p(-2) = 0$ and $p(4) = 0$. Setting the step
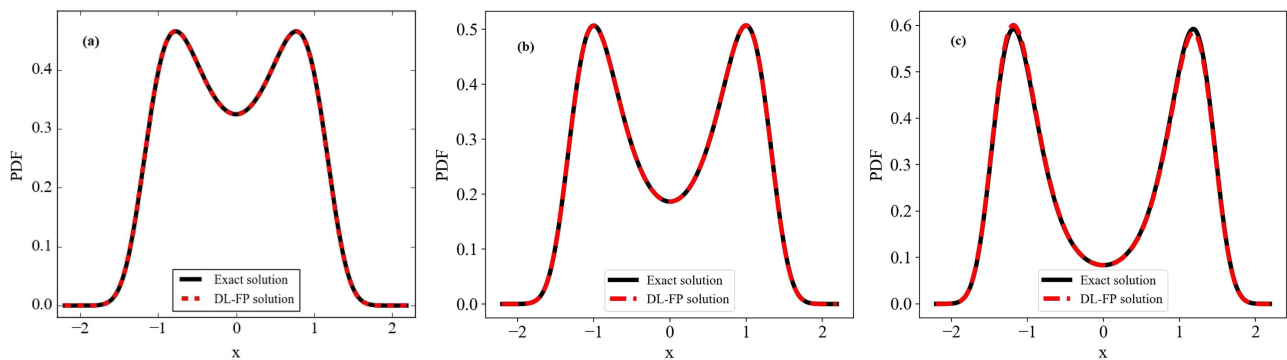

FIG. 2. The comparison results between the solution obtained from DL-FP algorithm and the exact solution for parameters $\beta = 0.5$, $\sigma = 0.5$. (a) $\alpha = 0.3$, (b) $\alpha = 0.5$, and (c) $\alpha = 0.7$. The red dashed line is the DL-FP solution, and the black solid line is the exact solution.
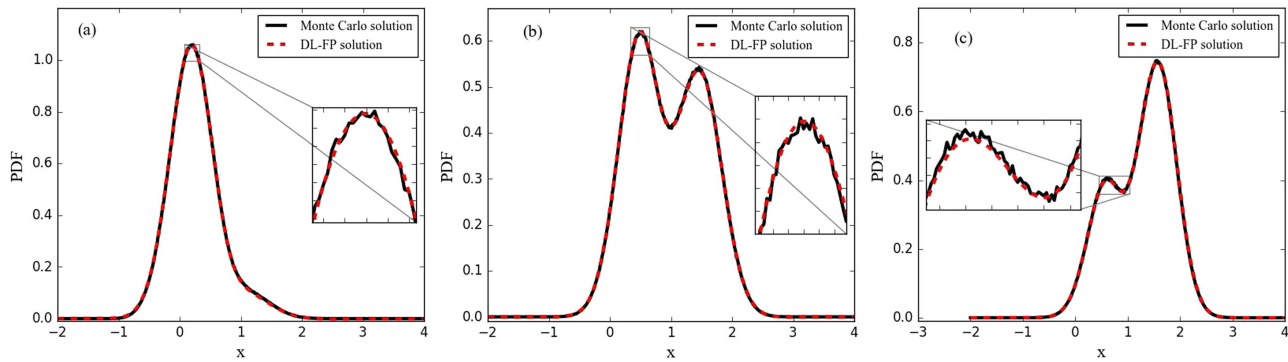
FIG. 3. The comparison results between the solutions obtained from the DL-FP algorithm and the Monte Carlo method with fixed noise intensity $\sigma = 0.5$ and different parameters for system (22). (a) $\alpha = 0.2$, (b) $\alpha = 0.5$, and (c) $\alpha = 0.6$. The red dashed line denotes the DL-FP solution, and the black solid line denotes the Monte Carlo solution.

length to be $\Delta x = 0.01$ to get the training set and the boundary set is $\{-2, 4\}$. With the DL-FP algorithm outlined above, the loss function is then minimized with the deep neural network which owns four layers and having 20 hidden nodes in every hidden layer. The activation function is selected as the tanh function, using the Adam method to minimize the loss function, and the minimum is about 10^{-6} after 5×10^4 iterations. Monte Carlo simulation is applied to get the PDF of Eq. (22). We adopt the second-order stochastic Runge-Kutta algorithm[39] to generate 10^8 data, then carry on the statistics to get the results which are shown in the black solid line in Fig. 3. Compared with the DL-FP solution, we find that the Monte Carlo solution is consistent with the DL-FP one. However, after zooming some local details, in contrast with the roughness of the Monte Carlo solution, the DL-FP result is rather smooth.

C. Example 3

The Duffing oscillator can be described as

$$\ddot{x} + \eta x + \alpha x + \beta x^3 = \sigma \Gamma, \tag{24}$$

where η is the damping coefficient, α and β denote the linear and nonlinear stiffness, respectively, and Γ represents a standard Gaussian white noise with σ being the noise intensity. Let

$$\begin{cases} \dot{x} = y, \\ \dot{y} = -\eta y - \alpha x - \beta x^3 + \sigma \Gamma. \end{cases} \tag{25}$$

Then, the corresponding stationary FP equation is

$$-\frac{\partial}{\partial x}[yp(x,y)] - \frac{\partial}{\partial y}\left[\left(-\eta y - \alpha x - \beta x^3\right)p(x,y)\right]$$
$$+ \frac{\sigma^2}{2}\frac{\partial^2}{\partial y^2}p(x,y) = 0, \tag{26}$$

and the exact steady-state solution is

$$p_s(x,y) = C \cdot \exp\left[-\frac{\eta}{\sigma^2}\left(y^2 + \alpha x^2 + \frac{\beta}{2}x^4\right)\right], \tag{27}$$

where C is the normalization constant.

Minimizing the loss function using the Adam optimization algorithm with 5×10^4 iterations, we get the results as is shown in Fig. 4. Here, we set in (26) to be $\alpha = 1.0$, $\beta = 0.2$, $\eta = 0.2$, $\sigma^2 = 0.1$. The input of the ANN is the discretization using the step of 0.01 with the range of $[-2, 2] \times [-2, 2]$. The training set owns 1.6×10^5 data, and the boundary set owns 1596 data. The boundary conditions are assumed to be zero for all the states.

Using the DL-FP method to solve Eq. (26), we set the penalty factors as $a_1 = 400$, $a_2 = 0.1$, $a_3 = 1$ based on the rules of the DL-FP method in 2D case and set the network with four hidden layers and 20 nodes in each hidden layer. The DL-FP solution is almost the same as the exact solution where the maximum absolute error in the JPDF is at the order 10^{-4}. The accuracy of the DL-FP solution can be seen more intuitively through the marginal probability density function (mPDF) as indicated in Figs. 4(d)–4(f). This example shows the validity of this algorithm in dealing with a single-degree-of-freedom Duffing system.

D. Example 4

Consider a 3D Ornstein-Uhlenbeck system as

$$\begin{cases} \dot{x}_1 = -\alpha x_1 + \sigma_1 \Gamma_1, \\ \dot{x}_2 = -\beta x_2 + \sigma_2 \Gamma_2, \\ \dot{x}_3 = -\gamma x_3 + \sigma_3 \Gamma_3, \end{cases} \tag{28}$$

where α, β, γ are constants, $\Gamma_i, i = 1, 2, 3$ are independent standard Gaussian white noises, and $\sigma_i, i = 1, 2, 3$ denote the corresponding noise intensity. Then, the reduced stationary FP equation is

$$\frac{\partial}{\partial x_1}\left(\alpha x_1 p(X)\right) + \frac{\partial}{\partial x_2}\left(\beta x_2 p(X)\right) + \frac{\partial}{\partial x_3}\left(\gamma x_3 p(X)\right) + \frac{\sigma_1^2}{2}\frac{\partial^2}{\partial x_1^2}p(X)$$
$$+ \frac{\sigma_2^2}{2}\frac{\partial^2}{\partial x_2^2}p(X) + \frac{\sigma_3^2}{2}\frac{\partial^2}{\partial x_3^2}p(X) = 0, \tag{29}$$

where $X = [x_1, x_2, x_3]$ and $p(X)$ is the joint PDF. The corresponding exact steady-state solution is

$$p_s(X) = \sqrt{\frac{\alpha\beta\gamma}{\pi^3\sigma_1^2\sigma_2^2\sigma_3^2}} \cdot \exp\left(-\frac{1}{2}\left(\frac{x_1^2}{\sigma_1^2} + \frac{x_2^2}{\sigma_2^2} + \frac{x_3^2}{\sigma_3^2}\right)\right). \tag{30}$$
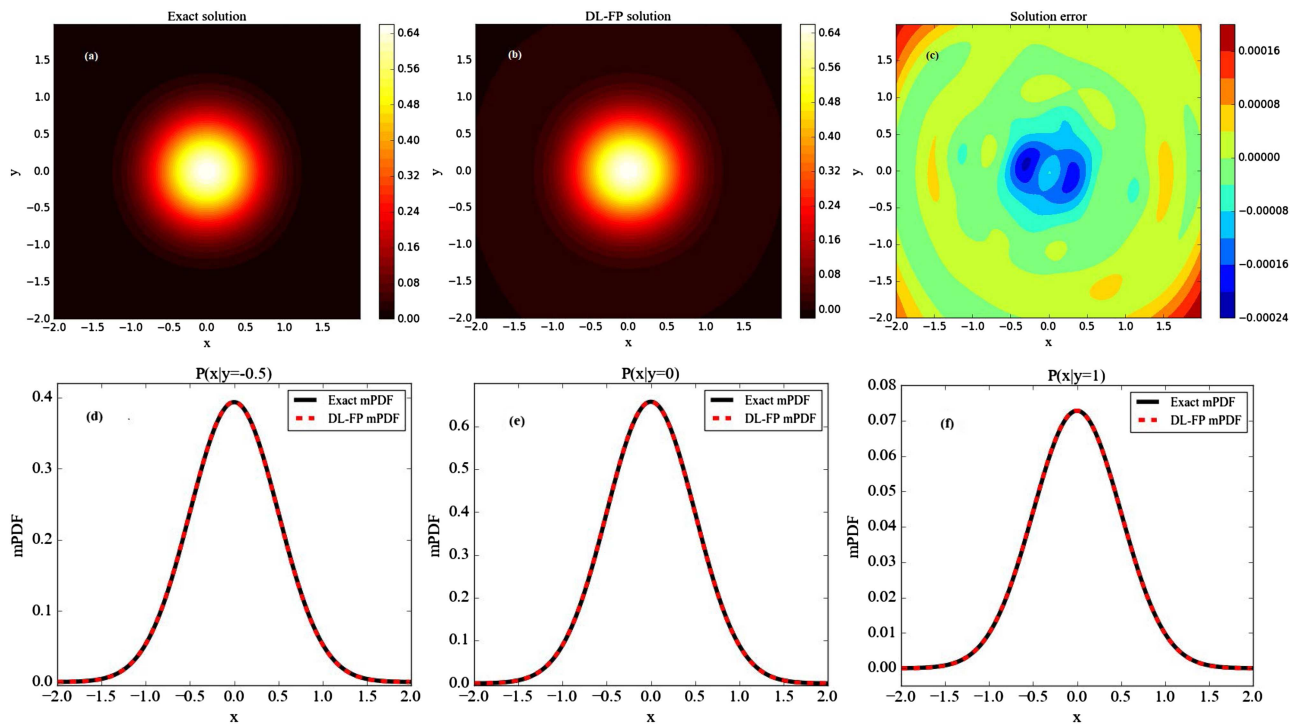
FIG. 4. The comparison results for Duffing system (24) with: $\alpha = 1.0$, $\beta = 0.2$, $\eta = 0.2$, $\sigma^2 = 0.1$. (a) The contour map of the exact JPDF, (b) the contour map of the DL-FP JPDF, (c) the error between the DL-FP solution and the exact solution, (d) the contrast between the exact marginal probability density (mPDF) and the DL-FP mPDF when $y = -0.5$, (e) the contrast between the exact mPDF and the DL-FP mPDF when $y = 0$, and (f) the contrast between the exact mPDF and the DL-FP mPDF when $y = 1$.

Based on the DL-FP method, we solve Eq. (29) with $\alpha = 0.5$, $\beta = 0.7$, $\gamma = 1$, $\sigma_1 = \sigma_2 = \sigma_3 = 0.25$. In Step 1, we select the training set in the range of $[-1, 1] \times [-1, 1] \times [-1, 1]$, then discretize the interval by 0.05 to get 6.4×10^4 data in the training set and also get 9128 data in the boundary set. In Step 2, we consider a network with seven hidden layers and 20 nodes in each hidden layer and set the penalty factor as $a_1 = 40$, $a_2 = 0.1$, $a_3 = 1$. Initializing the weight and bias of the neural network based on the ANN size and minimizing the loss function by the L-BFGS-B optimization algorithm with 5×10^5 iterations, we get the results in Fig. 5.

The DL-FP solution is almost the same as the exact solution where the maximum absolute error in the joint PDF is at the order of 10^{-2}. The error of the 2D mPDF is shown in Figs. 5(a)–5(c) with the order of 10^{-3}. The accuracy of the calculation is intuitively observed through the 1D mPDF in Figs. 5(d)–5(f). Such a high accuracy comes at the cost of huge calculations, for which the test is carried out in a Graphic Processing Unit (GPU) mode on a computer with a video card of Quadro k620 and the calculation time is 11 298.2 s. It is conceivable that the computation time is staggering as the dimensions increase and certainly this is due to the discretization of space.

IV. DISCUSSION

The DL-FP algorithm is proposed in this paper, and the high accuracy of the DL-FP algorithm can be observed in Sec. III

due to the particularity of the FP equation and the setting up of the network. In this section, we will discuss three problems, which are concerned in the proposed DL-FP algorithm: (1) the effect of the number of hidden layers on the calculated performances, (2) the importance of the penalty factors in the algorithm, and (3) how to select the optimization algorithm to train the FP equation.

A. The number of hidden layers

The influence of the network size on the accuracy is discussed first. For the selection of the optimization algorithm, then the Adam optimization algorithm is applied to illustrate the effect of the network size on the accuracy. We will discuss the 1D and 2D systems, respectively.

For the 1D system, we consider system (14) with fixed $\alpha = 0.3$, $\beta = 0.5$, $\sigma = 0.5$ as an example, where the network is assumed to have 1, 2, 3, 4, 5, and 6 hidden layers and 20 nodes in each hidden layer. We clearly observe in Fig. 6 that the loss function will reach the order of 10^{-7} when the hidden layer is 2 and 3 which is better than a single hidden layer in Fig. 7(a). However, the trend of the loss function gets worse when the number of the hidden layers exceeds 4. Specifically, when the number of the hidden layer is 6, it is difficult to train the network as $\mathcal{L}$ falling into the local optimum. So, the best number of the hidden layer is 2 or 3 and 20 nodes in each layer.
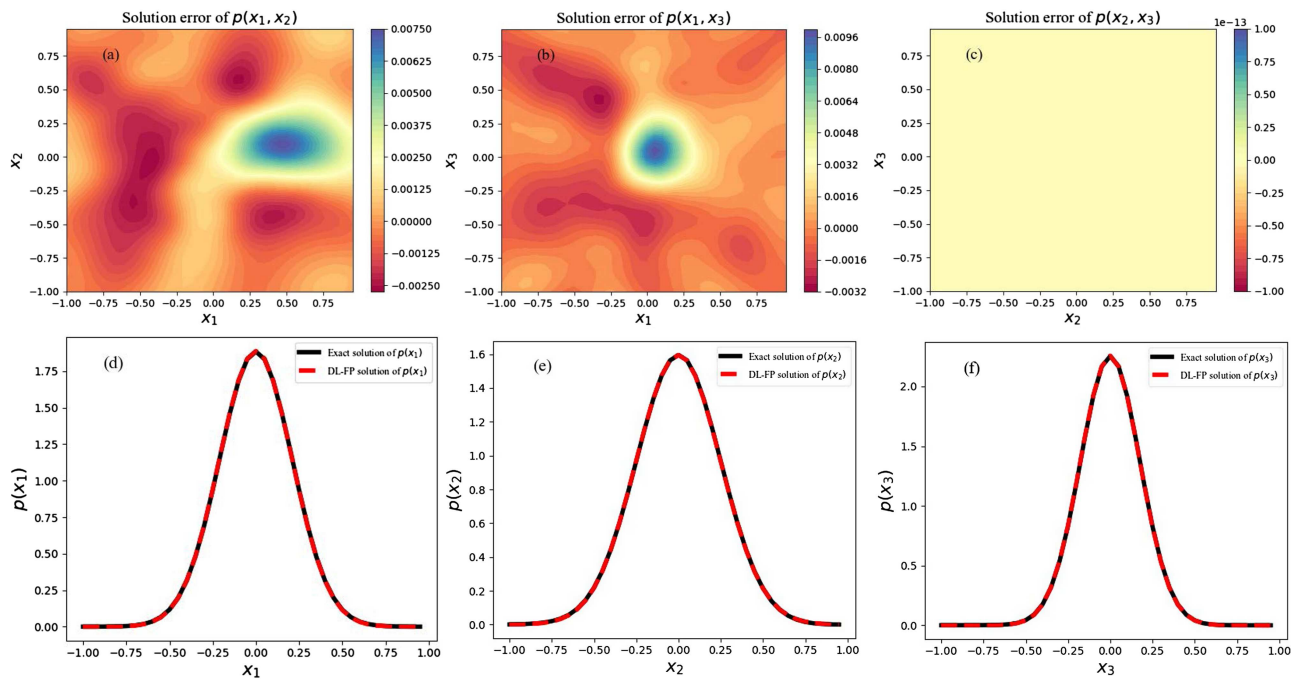
FIG. 5. The comparison results for system (28) with $\alpha = 0.5$, $\beta = 0.7$, $\gamma = 1$, $\sigma_1 = \sigma_2 = \sigma_3 = 0.25$. The error between the DL-FP solution and the exact solution for 2D mPDF. (a) $p(x_1, x_2)$, (b) $p(x_1, x_3)$, and (c) $p(x_2, x_3)$. The contrast between the exact solution and the DL-FP one for 1D mPDF. (d) $p(x_1)$, (e) $p(x_2)$, and (f) $p(x_3)$.

From Fig. 6(b), we can find that 2 hidden layers do not significantly improve the final accuracy of the calculation after a large number of iterations. Both 1 and 2 hidden layers are able to keep value of acc_{L_2} at around 0.996. However, more hidden layers do increase the speed to reach a high accuracy. A single hidden layer can be used to obtain

the higher accuracy under the Adam optimization algorithm since a deeper hidden layer is unnecessary.

For the 2D system, we take the Duffing system with $\alpha = 1.0$, $\beta = 0.2$, $\eta = 0.2$, $\sigma^2 = 0.1$ as an example and use the networks with 1, 2, 3, 4, and 5 hidden layers and 20 nodes in each hidden layer.
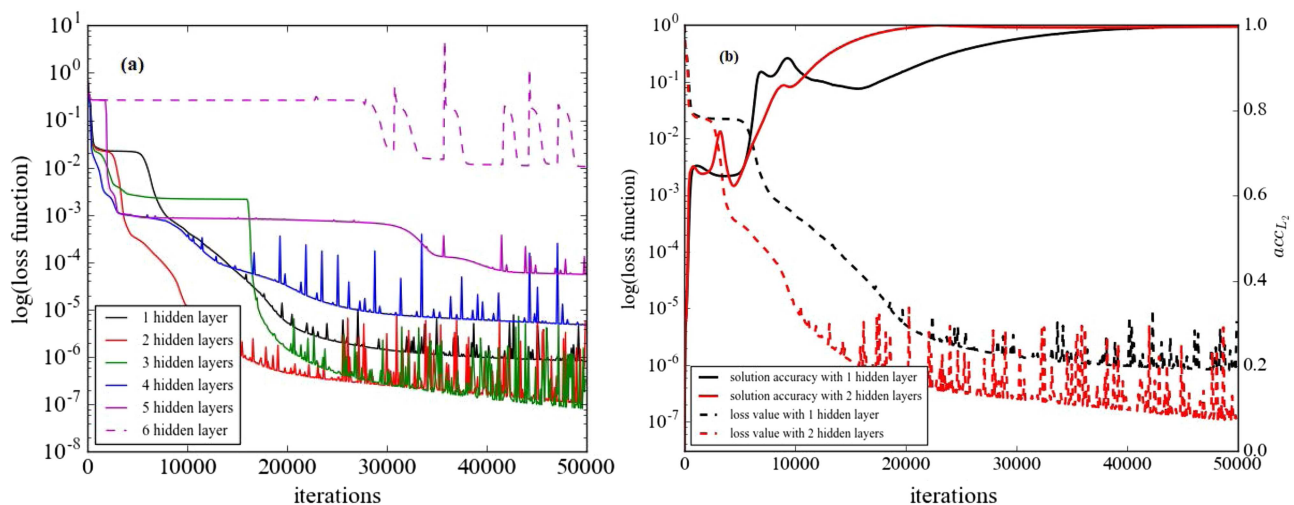

FIG. 6. (a) The comparison of loss functions for increasing network depth in terms of system (14). (b) The accuracy varies with the loss function in the double-well system. (The solid line represents the value of the loss function which corresponds to the left axis, and the dotted line represents the accuracy which corresponds to the right axis.)

FIG. 7. (a) The comparison of loss function for increasing network depth for Duffing system (24). (b) The accuracy varies with the loss function in (24) (The solid line represents the value of the loss function which corresponds to the left axis, and the dotted line represents the accuracy which corresponds to the right axis.)
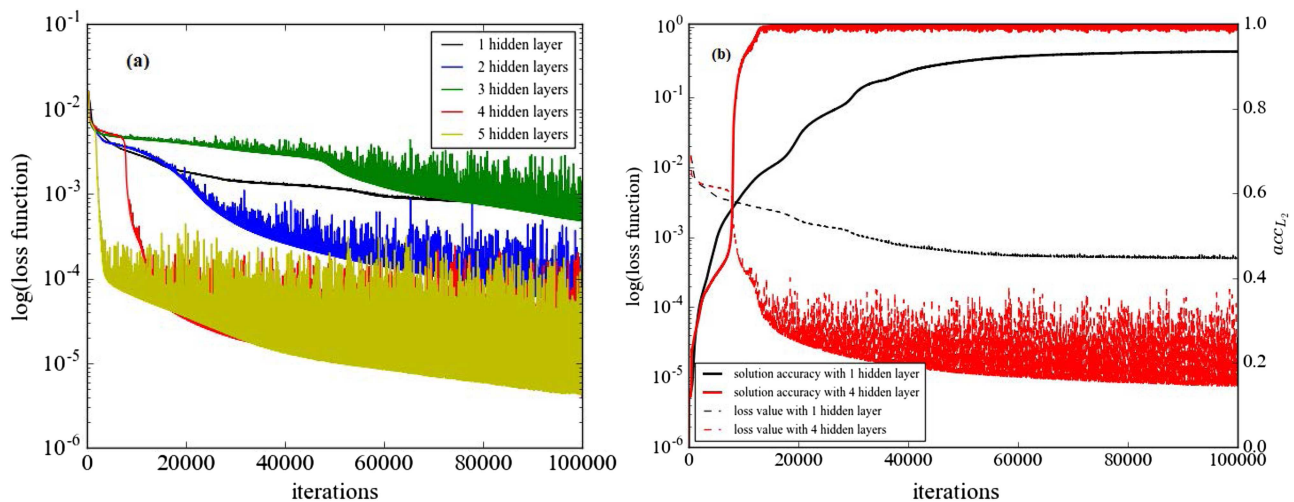
The results are given in Fig. 7. Different from the 1D case, the influence of more hidden layers on the accuracy of calculation and the optimization are more advantageous than a single hidden layer. In Fig. 7(a), with an increase of the number of hidden layers, the loss function decreases more rapidly. When there are 5 hidden layers, it will quickly reach the order of 10^{-5}, while the single hidden layer can only reach the order of 10^{-3}. The multihidden layers work better for the improvement of accuracy in Fig. 7(b). The accuracy of a single hidden layer is maintained at about 0.94, while the 4 hidden layers can reach about 0.99. So, in 2D computation, it is very helpful to use more hidden layers to improve the speed of training and the accuracy.

B. The penalty factor

This section is to discuss the effect of a penalty factor in the 2D case and take Duffing system (24) as an example. We select the interval to be $[-2, 2] \times [-2, 2]$ and $\Delta x = 0.05$, $\alpha = 1.0$, $\beta = 0.2$, $\eta = 0.2$, $\sigma^2 = 0.1$. We use an ANN with 5 hidden layers where the nodes in each hidden layer are 20.

We set the penalty factor to be $a_1 = 1$, $a_2 = 1$, $a_3 = 1$, which is equal to the traditional mean square loss function. After training 1×10^5 times with the Adam algorithm, the result of the loss function is about 1.52×10^{-7}, and the absolute error between the DL-FP solution and the exact solution is shown in Fig. 8(a). We observe that the solution error is very small at the boundary, but it is large in the
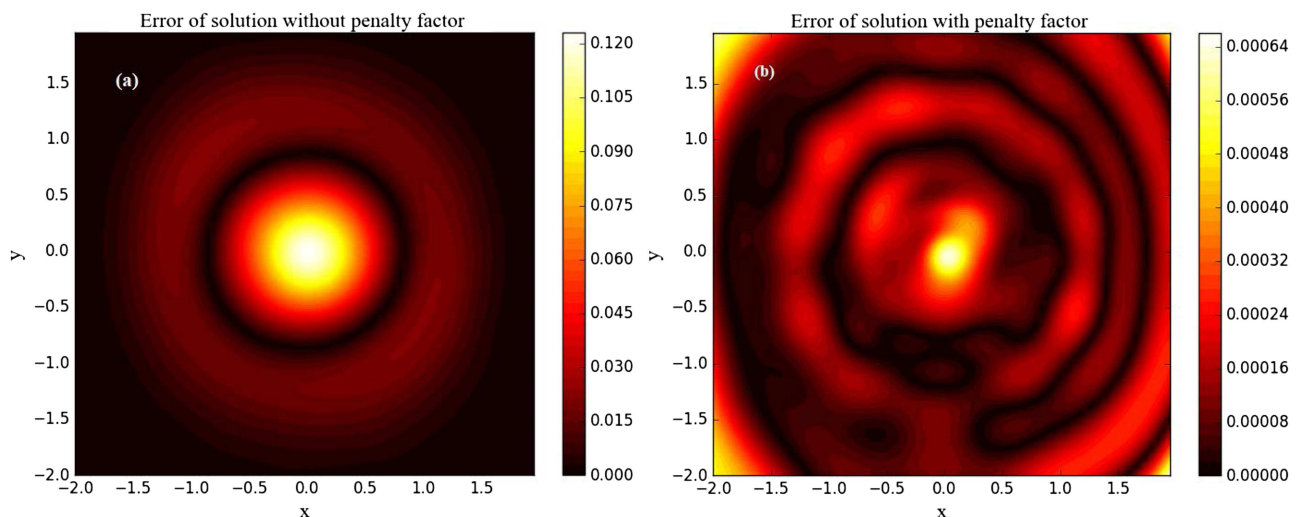

FIG. 8. Influences of the penalty factors on the absolute error of solution for Duffing system (24). (a) Without penalty factors and (b) with penalty factors.
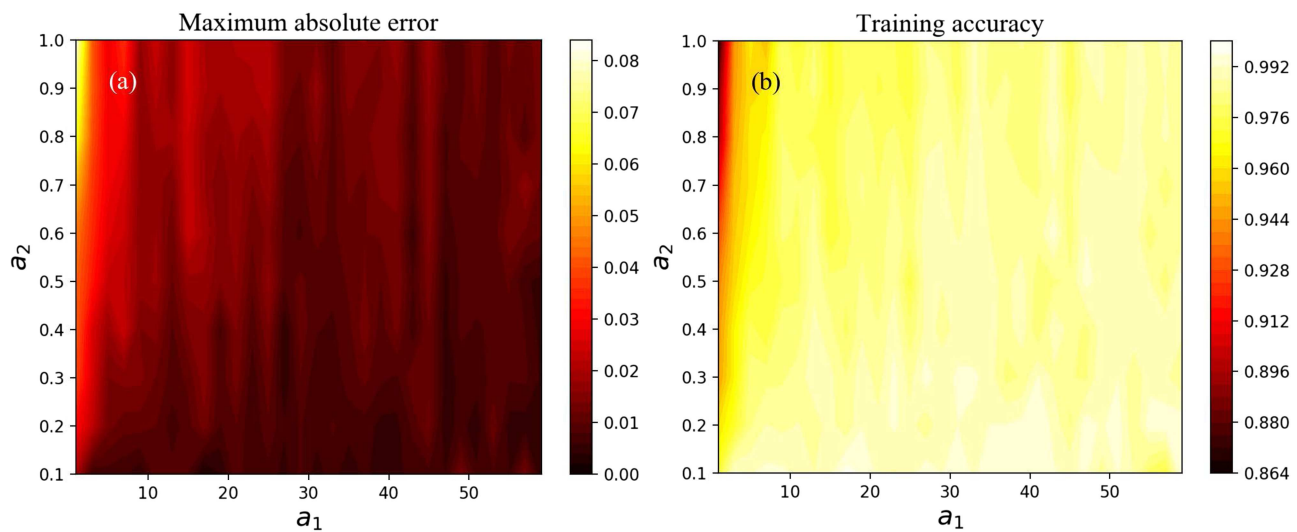
FIG. 9. Influences of the penalty factors a_1 and a_2 on the maximum absolute error of the solution for Duffing system (24) applying the Adam method. (a) Maximum absolute error and (b) training accuracy.

middle part. Because the trial solution is easy to satisfy the boundary condition and the supervision training on the form of the equation is not strict enough, which lead to the training failure. So, we increase the punishment of Eq. (11). According to the rules in Sec. II, we set the penalty factor to be $a_1 = 80$, $a_2 = 0.1$, $a_3 = 1$. After training 1×10^5 times applying the Adam method, the result of loss function is 1.52×10^{-6} and the absolute error between the DL-FP and the exact solutions is shown in Fig. 8(b). Hence, the solution error is about the order of 10^{-4}. When the weight of Eq. (11) increases,

the error caused by each data point in the training will be enlarged. Besides, the boundary condition is easily satisfied so that the training is successful. Therefore, it is very necessary to add the penalty factor in the loss function.

System (24) is taken as an example to illustrate the relationship between the accuracy and the penalty factors. The parameters of system (24) are kept the same, except that the discrete interval is changed to 0.2 and $N_p = 40$. We consider a deep neural network with four hidden layers and 20 nodes in each hidden layer and set the
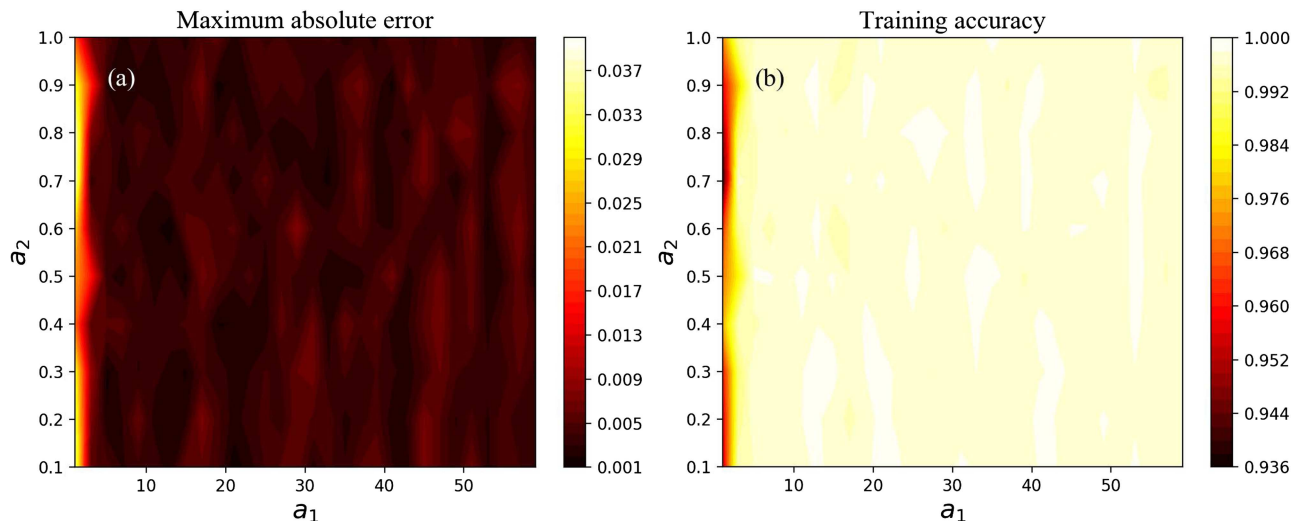

FIG. 10. Influences of the penalty factors a_1 and a_2 on the maximum absolute error of the solution for Duffing system (24) adopting the L-BFGS-B algorithm. (a) Maximum absolute error and (b) training accuracy.

TABLE II. The comparison of Adam and L-BFGS-G algorithms in the 1D case.

Number of hidden layers	Method	Loss value	acc_{L_2}	Maximum absolute error	Time cost (s)
1	Adam	2.96×10^{-6}	0.9988	6.7×10^{-4}	47.2
	L-BFGS-B	4.38×10^{-6}	0.961	2.12×10^{-2}	33.7
2	Adam	1.03×10^{-7}	0.9962	1.09×10^{-3}	78.4
	L-BFGS-B	2.56×10^{-7}	0.9989	5.6×10^{-4}	26.8
3	Adam	7.93×10^{-8}	0.9979	1.09×10^{-3}	110.6
	L-BFGS-B	9.03×10^{-8}	0.9971	1.54×10^{-3}	32.9
4	Adam	4.61×10^{-6}	0.9368	3.4×10^{-2}	134.6
	L-BFGS-B	5.63×10^{-7}	0.9976	1.28×10^{-3}	25.7

TABLE III. The comparison of Adam and L-BFGS-B algorithms in the 2D case.

Number of hidden layers	Method	Loss value	acc_{L_2}	Maximum absolute error	Time cost (s)
1	Adam	6.60×10^{-4}	0.9056	7.69×10^{-2}	1249.9
	L-BFGS-B	2.49×10^{-3}	0.7134	2.36×10^{-1}	149.7
2	Adam	4.89×10^{-5}	0.9907	7.21×10^{-3}	2343.5
	L-BFGS-B	4.98×10^{-6}	0.9980	4.46×10^{-3}	240.4
3	Adam	1.27×10^{-5}	0.9977	3.36×10^{-3}	3054.4
	L-BFGS-B	9.92×10^{-6}	0.9974	5.88×10^{-3}	183.1
4	Adam	4.00×10^{-6}	0.9993	1.7×10^{-2}	3985.3
	L-BFGS-B	5.90×10^{-6}	0.9978	5.26×10^{-3}	264.4
5	Adam	4.27×10^{-6}	0.9993	6.45×10^{-4}	4793.6
	L-BFGS-B	1.27×10^{-6}	0.9985	3.42×10^{-3}	353.0

penalty factors as $a_3 = 1$, $a_1 \in [0.1, 1]$ and $a_2 \in [1, 60]$. The variation of the maximum absolute error and accuracy with penalty factors a_1 and a_2 under Adam and L-BFGS-B algorithms is discussed in detail. The number of iterations for both methods is 5×10^4.

First, the Adam algorithm is used for optimization and the results are indicated in Fig. 9. The maximum absolute error and training accuracy are not ideal when both a_1 and a_2 approach 1. But when we increase a_1 and decrease a_2 gradually, the training effect improves a lot. Therefore, when the Adam algorithm is considered to train the model, the training effect is strongly dependent on the penalty factors.

Second, optimized by the L-BFGS-B method, the results are shown in Fig. 10. We find that the dependence of the L-BFGS-B method on the penalty factors is weaker than that in the Adam algorithm. However, the training effect is still not good as a_1 approaches 1, so increasing a_1 is very helpful for the training effect.

Therefore, either the Adam or the L-BFGS-B algorithm is adoptable by increasing a_1 and decreasing a_2, which plays a positive role in the training effect. In this paper, we set $a_1 = N_p$ and $a_2 = 0.1$ to be conservative. In the actual training, as long as a_1 is appropriately increased, and a_2 is decreased, the network training can be promoted.

C. The optimization algorithm

Selecting the appropriate optimization algorithm is very important during training the network. The Adam optimization algorithm is adopted in the discussion of Sec. IV A. It was found that a relatively good training can only be achieved when the iteration reaches 5×10^4, and the training time of this method will reach 1 h in the 2D case which is too long to be accepted for a fast operation. Hence, we try to improve the optimization algorithm. It is easy to find that the gradient disappears if the stochastic gradient descent method and momentum method are adopted. We find that the L-BFGS-B algorithm[36] of the quasi-Newton method is very effective. We compare the calculation of the Adam and the L-BFGS-B methods in the 1D and 2D cases, and the results are shown in Tables II and III. All the tests are carried out in a GPU mode on a computer with a video card of Quadro k620.

First, we take system (14) as an example. Set the interval to be $[-2.3, 2.3]$ and the parameters to be $\Delta x = 0.01$, $\alpha = 0.3$, $\beta = 0.5$,

$\sigma = 0.5$. Both methods set the number of iteration to be 5×10^4 and the number of nodes in each layer of the network to be 20. The results of the comparison of the calculation with different numbers of hidden layers are shown in Table II. We find that the L-BFGS-B method has a significant kicking effect on the calculation time which can ensure that the calculation can be completed in about 30 s, and the effect of multihidden layers is better than that of the single hidden layer. Additionally, the accuracy of the training is better when the value of $\mathcal{L}$ reaches the order of 10^{-7} in both the Adam and L-BFGS-B algorithms. The Adam algorithm will overfit with the increase of the number of hidden layers, but the L-BFGS-B algorithm will not.

In the 2D case, we take system (24) as a study example. Set the interval to be $[-2, 2] \times [-2, 2]$ and the parameters to be $\Delta x = 0.05$, $\alpha = 1.0$, $\beta = 0.2$, $\eta = 0.2$, $\sigma^2 = 0.1$. The network is set up in the same way as in the 1D case. It can be clearly seen in Table III that the L-BFGS-B algorithm has great advantages concerning computing time. Moreover, when the value of the loss function reaches the order of 10^{-6}, the training accuracy is very high. Whether using the Adam or the L-BFGS-B method, the accuracy will increase with the number of layers. Therefore, the L-BFGS-B technique can achieve the same effect as the Adam method but saves a lot of time.

V. CONCLUSIONS

This paper proposed a numerical method named DL-FP to solve the FP equation using deep learning. Four examples of 1D, 2D, and 3D systems have illustrated that the proposed DL-FP algorithm has a high accuracy. Compared with the traditional numerical scheme to solve the FP equation, the DL-FP algorithm does not require an interpolation or other reconstruction techniques that are different from the finite element method and the finite difference algorithm. The obtained solution is unnecessary to ask any coordinate transformation but more smooth than the Monte Carlo simulation. Inspired by the traditional solution to differential equations by ANNs, in this paper, the normalization condition is introduced as a supervision condition, so as to avoid the situation that the training solution of FP equation is zero.

Due to the introduction of normalized monitoring conditions, a penalty factor is given for the loss function (using the traditional mean square error format) effectively trained to minimize the calculation. We introduce a penalty factor to solve this problem effectively. In particular, this paper gives rules to set the penalty factors in 1D and 2D cases. In Sec. IV B, Duffing system (24) is adopted as an example. However, the setting rules of such penalty factors still need to be studied in higher dimensions.

Besides, we discuss the setting of the number of hidden layers and the optimization algorithm in ANN. In the setting of the network, we suggest to choose deeper hidden layers to the higher accuracy. In terms of the number of iterations needed for training, we suggest that it is more reasonable to determine the training number according to the precision of the loss function. In the 1D case, the loss function reaches the order of 10^{-7}, and in the 2D case, it reaches the order of 10^{-6}. In the selection of algorithms for optimizing loss functions, we suggest the Adam and the L-BFGS-B algorithms. In the 1D case, there is little difference between them, but in the 2D case, the L-BFGS-B algorithm strongly shortens the training time.

It is important to point out that, although the DL-FP algorithm can be used to deal with all solutions of the high-dimensional FP equation in the region, the discretization method will inevitably lead to the problem of data explosion and the increase of parameters with the dimension increasing. Therefore, when solving a higher-dimensional FP equation, it needs a higher performance and a higher storage capacity, otherwise, one should reduce the dimension of FP equation to reduce the amount of data used for training.

REFERENCES

[1] Z. Wang, Y. Xu, and H. Yang, "Lévy noise induced stochastic resonance in an FHN model," Sci. China Technol. Sci. **59**, 371–375 (2016).

[2] Y. Li, Y. Xu, J. Kurths, and X. Yue, "Lévy-noise-induced transport in a rough triple-well potential," Phys. Rev. E **94**, 042222 (2016).

[3] M. Escobedo, M. A. Herrero, and J. J. L. Velazquez, "Radiation dynamics in homogeneous plasma," Physica D **126**, 236–260 (1999).

[4] A. Ceccato and D. Frezzato, "Remarks on the chemical Fokker-Planck and Langevin equations: Nonphysical currents at equilibrium," J. Chem. Phys. **148**, 064114 (2018).

[5] B. M. Grafov, "Fokker-Planck equations for stochastic diffusion associated with Markovian electrochemical noise," Russ. J. Electrochem. **51**, 278–280 (2015).

[6] H. Karcher, S. Lee, M. Kaazempurmofrad, and R. Kamm, "A coarse-grained model for force-induced protein deformation and kinetics," Biophys. J. **90**, 2686–2697 (2006).

[7] Y. Xu, J. Feng, J. Li, and H. Zhang, "Lévy noise induced switch in the gene transcriptional regulatory system," Chaos **23**, 013110 (2013).

[8] I. S. M. Fokou, C. N. D. Buckjohn, M. S. Siewe, and C. Tchawoua, "Probabilistic behavior analysis of a sandwiched buckled beam under Gaussian white noise with energy harvesting perspectives," Chaos Solitons Fractals **92**, 101–114 (2016).

[9] G. Furioli, A. Pulvirenti, E. Terraneo, and G. Toscani, "Fokker-Planck equations in the modeling of socio-economic phenomena," Math. Models Methods Appl. Sci. **27**, 115–158 (2017).

[10] J. Elgin, "The Fokker-Planck equation: Methods of solution and applications," Opt. Acta Int. J. Opt. **31**, 1206–1207 (1996).

[11] J. Náprstek and R. Král, "Finite element method analysis of Fokker-Planck equation in stationary and evolutionary versions," Adv. Eng. Softw. **72**, 28–38 (2014).

[12] R. F. Galán, E. G Bard, and N. N. Urban, "Stochastic dynamics of uncoupled neural oscillators: Fokker-Planck studies with the finite element method," Phys. Rev. E **76**, 056110 (2007).

[13] Y. Jiang, "A new analysis of stability and convergence for finite difference schemes solving the time fractional Fokker-Planck equation," Appl. Math. Model. **39**, 1163–1171 (2015).

[14] B. Sepehrian and M. K. Radpoor, "Numerical solution of non-linear Fokker-Planck equation using finite differences method and the cubic spline functions," Appl. Math. Comput. **262**, 187–190 (2015).

[15] A. N. Drozdov, "Accurate path integral representations of the Fokker-Planck equation with a linear reference system: Comparative study of current theories," Phys. Rev. E **57**, 146–158 (1998).

[16] M. Chandrika and Y. N. Kaznessis, "Path-integral method for predicting relative binding affinities of protein-ligand complexes," J. Am. Chem. Soc. **131**, 4521–4528 (2009).

[17] Y. Xu, W. Zan, W. Jia, and J. Kurths, "Path integral solutions of the governing equation of SDEs excited by Lévy white noise," J. Comput. Phys. **394**, 41–45 (2019).

[18] J. Biazar, P. Gholamin, and K. Hosseini, "Variational iteration method for solving Fokker-Planck equation," J. Franklin Inst. **347**, 1137–1147 (2010).

[19] M. Torvattanabun and S. Duangpithak, "Numerical simulations of Fokker-Plank equation by variational iteration method," Int. J. Math. Anal. **5**, 2193–2201 (2011).

[20] Z. Sun, J. Carrillo, and C. W. Shu, "A discontinuous Galerkin method for nonlinear parabolic equations and gradient flow problems with interaction potentials," J. Comput. Phys. **352**, 76–104 (2017).

[21] Y. Zheng, C. Li, and Z. Zhao, "A fully discrete discontinuous Galerkin method for nonlinear fractional Fokker-Planck equation," Math. Probl. Eng. **2010**, 279038 (2010).

[22] I. Norio and G. Kosuke, "Thermal fluctuations and stability of a particle levitated by a repulsive Casimir force in a liquid," Phys. Rev. E **88**, 052133 (2013).

[23] E. Hirvijoki, T. Kurki-Suonio, S. Äkäslompolo, J. Varje, T. Koskela, and J. Miettunen, "Monte Carlo method and high performance computing for solving Fokker-Planck equation of minority plasma particles," J. Plasma Phys. **81**, 435810301 (2015).

[24] D. Barrow and N. Kourentzes, "The impact of special days in call arrivals forecasting: A neural network approach to modelling special days," Eur. J. Oper. Res. **264**, 967–977 (2018).

[25] M. Schirmer, M. Lehning, and J. Schweizer, "Statistical forecasting of regional avalanche danger using simulated snow-cover data," J. Glaciol. **55**, 761–768 (2017).

[26] G. Cybenko, "Approximation by superpositions of a sigmoidal function," Math. Control Signals Syst. **2**, 303–314 (1989).

[27] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," IEEE Trans. Neural Netw. **9**, 987–1000 (1998).

[28] T. Leephakpreeda, "Novel determination of differential-equation solutions: Universal approximation method," J. Comput. Appl. Math. **146**, 443–457 (2002).

[29] B. P. van Milligen, V. V. Tribaldos, and J. A. Jiménez, "Neural network differential equation and plasma equilibrium solver," Phys. Rev. Lett. **75**, 3594 (1995).

[30] R. S. Beidokhti and A. Malek, "Solving initial-boundary value problems for systems of partial differential equations using neural networks and optimization techniques," J. Franklin Inst. **346**, 898–913 (2009).

[31]A. Malek and R. S. Beidokhti, "Numerical solution for high order differential equations using a hybrid neural network-optimization method," Appl. Math. Comput. **183**, 260–271 (2006).

[32]J. Berg and K. Nyström, "A unified deep artificial neural network approach to partial differential equations in complex geometries," Neurocomputing **317**, 28–41 (2018).

[33]M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," J. Comput. Phys. **378**, 686–707 (2019).

[34]J. Han, A. Jentzen, and E. Weinan, "Solving high-dimensional partial differential equations using deep learning," Proc. Natl. Acad. Sci. U.S.A. **115**, 8505–8510 (2018).

[35]J. Sirignano and K. Spiliopoulos, "DGM: A deep learning algorithm for solving partial differential equations," J. Comput. Phys. **375**, 1339–1364 (2018).

[36]C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization," ACM Trans. Math. Softw. **23**, 550–560 (1997).

[37]J. Ma, Y. Xu, Y. Li, R. Tian, and J. Kurths, "Predicting noise-induced critical transitions in bistable systems," Chaos **29**, 081102 (2019).

[38]J. Ma, Y. Xu, W. Xu, Y. Li, and J. Kurths, "Slowing down critical transitions via Gaussian white noise and periodic force," Sci. China Technol. Sci. **62**, 2144–2152 (2019).

[39]R. L. Honeycutt, "Stochastic Runge-Kutta algorithms. I. White noise," Phys. Rev. A **45**, 600 (1992).